Noname manuscript No.
(will be inserted by the editor)

Ant Clustering with Locally Weighted Ant Perception and Diversified Memory

Gilbert L. Peterson · Christopher B. Mayer · Thomas L. Kubler

the date of receipt and acceptance should be inserted later

Abstract Ant clustering algorithms are a robust and flexible tool for clustering data that have produced some promising results. This paper introduces two improvements that can be incorporated into any ant clustering algorithm: kernel function similarity weights and a similarity memory model replacement scheme. A kernel function weights objects within an ant's neighborhood according to the object distance and provides an alternate interpretation of the similarity of objects in an ant's neighborhood. Ants can hill-climb the kernel gradients as they look for a suitable place to drop a carried object. The similarity memory model equips ants with a small memory consisting of a sampling of the current clustering space. We test several kernel functions and memory replacement schemes on the Iris, Wisconsin Breast Cancer, and Lincoln Lab network intrusion datasets. Compared to a basic ant clustering algorithm, we show that kernel functions and the similarity memory model increase clustering speed and

 This paper was supported by the Air Force Office of Scientific Research.

Gilbert L. Peterson
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433
E-mail: gilbert.peterson@afit.edu

Christopher B. Mayer
Department of Electrical and Computer Engineering
United States Naval Academy
M.S. 14B
105 Maryland Avenue
Annapolis, MD 21402
E-mail: cmayer@usna.edu

Thomas L. Kubler
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433
E-mail: thomas.kubler@afit.edu

cluster quality, especially for datasets with an unbalanced class distribution, such as network intrusion.

1 Introduction

Computer-based ant clustering algorithms are inspired by the ability of real ants to organize their nests [6, 7]. In computer-based ant clustering algorithms, a colony of ants group individual data objects that have been dispersed randomly onto a two-dimensional grid. The ants spatially sort these objects by stochastically picking up and dropping data objects. The probability of picking up an object increases in direct proportion to how dissimilar an object is to others in its neighborhood. Conversely, the probability of dropping an object increases in proportion to how similar the object an ant is carrying is to objects near the ant's current position. Many iterations of picking up and dropping items results in clusters of data objects in which similar objects are in the same cluster.

Clustering performance can be evaluated based on the speed of the clustering and on the quality of the clustering. The speed of clustering evaluation consists of the runtime to achieve the clustering results often measured in the number of iterations when using ant clustering algorithms. For determing the quality of the clustering, we evaluate in two ways. The first compares the number of created clusters with the number of true classes present in the data. The second is a calculation of the similarity of objects in a cluster. A good clustering algorithm minimizes the intra-cluster variance while maximizing inter-cluster variance.

Using Adaptive Time-dependent Transporter Ants (ATTA) [17] as a basis for a simple ant clustering algorithm, this paper presents two new techniques that increase clustering speed, reduce the number of clusters, and shrink cluster variances. These techniques could be incorporated into most ant clustering algorithms. While we achieve notable performance gains, we do note that additional research will be required to see how these improvements compare to advanced memory and ant perception techniques.

The first improvement is the use of locally weighted regression kernel functions [1]. Kernel functions are used in the calculation of how similar (or dissimilar) an object is to its neighboring objects. Kernel functions weight the influence that objects have upon each other relative to how far apart they are in the grid. In a sense, the kernel weightings produce a kind of gradient map of the objects in an ant's neighborhood. Compared to algorithms in which objects are equally weighted, a kernel function can provide an alternate interpretation of the similarity (or dissimilarity) of objects in a neighborhood. We make use of the kernel function in two ways. First, kernel functions are used to determine whether an object should be picked up or dropped. Second, we give ants the ability to hill-climb cluster gradients so that they have a higher probability of dropping objects near cluster centers. This paper compares and contrasts the effects of several kernel functions on clustering speed and quality.

The second major improvement is the introduction of a similarity-based memory system. In the Similarity Memory Model (SiMM), ants maintain a small memory of recently placed objects. After picking up an object, an ant has a small chance to jump directly to the location where the most similar object in its memory was dropped. This lets ants avoid the inefficiency of a random walk. Upon dropping an object, the dropped object replaces the most similar object in the ant's memory. Replacing objects in memory based on similarity maintains a memory that can capture the nuances of

class distributions rather than reflecting the sampling distribution. Thus, the Similarity Memory Model better remembers the sample space, and not just the sampling from that space. This results in less loss of information about low-frequency sampled areas of the decision space. We experiment with individual and shared ant memories and find that there is little performance difference between the two.

The kernel functions and Similarity Memory Model are implemented as part of a new algorithm called Ant Clustering with Kernel Functions and Similarity Memory Model (AntKSiMM). Experimental evaluation shows that the AntKSiMM algorithm speeds clustering and improves the clustering quality of large complex datasets.

This paper is organized as follows. Section 2 contains relevant related work. Section 3 describes a basic ant clustering algorithm based on ATTA. Next in Section 4 is a detailed explanation of our kernel and memory enhancements to the basic algorithm and introduces the AntKSiMM algorithm. Section 5 analyzes the performance of AntKSiMM on three datasets: the Iris, the Wisconsin Breast Cancer and the Lincoln Lab Intrusion Detection System. Finally, Section 6 concludes by summarizing the significance of the AntKSiMM approach.

2 Related Work

Data clustering seeks to uncover patterns in a set of objects by partitioning objects (data) into groups called clusters based on properties of the objects. Ideally, highly similar objects are clustered together and different clusters have dissimilar members. Once the objects have been clustered, typical next steps are to infer a rule set for cluster membership or to use the clusters themselves as the basis for establishing classes (i.e., each cluster is a class and the objects in the cluster belong to the class). In cases where objects have been assigned classes already, a good clustering algorithm should produce the same number of clusters as there are classes and incorrectly cluster very few objects.

Over the years several clustering algorithms have been proposed. Fast algorithms such as k-means [26] are sometimes impractical because their clustering quality depends on knowing the number of clusters, k, beforehand. The g-means [13] and χ-means [31] algorithms and Bayesian clustering algorithms such as AutoClass [2] can identify the number of clusters, but pay a penalty in terms of run times. Incremental clustering algorithms such as Cobweb [9] and Classit [11] consider objects one-by-one, forming new clusters or adding to existing clusters depending on how the new object impacts cluster quality. However, the order in which objects are presented can have a major impact on clustering quality from run to run. Genetic clustering algorithms such as [4, 21], like the incremental clustering algorithms, employ a global cluster quality evaluation function which can be costly to compute.

In an attempt to build a better clustering algorithm that is more or less free of the drawbacks of previous techniques, researchers have proposed several clustering algorithms that mimic the ability of natural ants to cluster items in their nests. Like their natural counterparts, ant clustering algorithm ants cluster objects even though each ant has a small number of decision rules, limited perception, and communicates only through the environment. An overview of ant and swarm clustering including research directions can be found in [19].

Deneubourg et al. [6] proposed the first ant clustering algorithm as a demonstration of ant clustering behavior. In their algorithm, a colony of ant agents randomly move

about a two-dimensional grid upon which objects are initially scattered. When an ant encounters an object, it picks it up with a probability that increases as the number of similar objects in the area around the ant decreases. Again walking randomly about the grid, an ant, upon encountering a free cell, drops the object with a probability that increases as the density of the objects in the ant's neighborhood increases. In this fashion, and after many pick-up and drop operations, similar objects are clustered together.

Lumer and Faieta [24] created the first ant clustering algorithm for data clustering by changing the pick-up and drop probability calculations so that they consider object similarity in addition to object density. A notable first, their algorithm suffered from slow convergence and high numbers of clusters. Since then, ant memories and adjustments to ant perception have been suggested as ways to increase clustering speed and quality. An overview of these topics is presented in the following subsections.

2.1 Ant Memories

Small short-term ant memories were introduced in [24] in order to speed convergence. Upon picking up an object, the ant searches its memory for the most similar object that it has carried before and uses the location where the similar object was dropped as a guide for dropping its currently held object [3, 17]. In these "look-ahead" memory schemes, with some probability the ant jumps directly to the best location or performs the normal random walk in search of a suitable drop location. Comparing individual ant memories against a shared colony-wide memory was found to have little effect on clustering performance [29]. Instead, [29] found that communicating the states of the clusters had the biggest impact on performance, but makes the algorithm further removed from primitive ants. Another downside to small short-term memories is that they are easily biased by the recent past, especially if a queue-based (FIFO) replacement policy is used.

Inspired by both [24] and [17], we introduce a similarity based memory for ant clustering algorithms that overcomes the biases of past ant memory systems while remaining simple and locally based.

2.2 Ant Perception

Modification to the ant's perception of its neighborhood has been proposed as a way to boost performance. Modifying ant pick-up and drop behavior based on the ant's neihborhood as being either "dense", "sparse", or "empty" was used in [3]. The idea of using alternating rounds of ant clustering and k-means along with the idea of moving clusters instead of individual objects in later rounds was proposed in [28] as a way to speed convergence. While the speed of clustering increased and the number of clusters was acceptable, the clustering of the individual objects depends more on k-means than the ants that move entire clusters of objects.

Handl et al. make several modifications to ant perception in their Adaptive Time-dependent Transporter Ants (ATTA) algorithm [17]. First, ants become less sensitive to object similarities (less discriminating) the longer they fail to pick-up objects and, conversely, more discriminating if they have a high rate of picking up objects. As a result, the ants learn the appropriate level of similarity sensitivity for the dataset,

which is captured in the adaptive α component. Second, they make changes to the similarity function and pick-up and drop probability equations. The changes cause high dissimilarities to always trigger a pick-up and for more similar items to be picked up with a probability inversely proportional to the square of the dissimilarity. The opposite applies to dropping an item; highly similar items are always dropped while less similar items are dropped with probability that is directly proportional to the dissimilarity score raised to the fourth power. Dissimilarity scores have the range of $[0, 1]$ where dissimilar objects tend toward 1 and similar toward 0. Third, [17] employs a "time dependent modulation of the neighborhood function". At the start of the algorithm, ants normalize object similarity scores by the number of grid cells in the ant's vision radius, as is done in most ant clustering algorithm algorithms since [24]. For a short time in the middle of a run, the number of occupied cells in the ant's vision radius becomes the normalizing factor. This switch causes the ants to enlarge the clusters by spreading the objects out rather than bringing them together. The final modification is the inclusion of an increasing radius of perception, as the number of iterations increases, the perception radius linearly increases.

The combination of the fuzzy C-means algorithm and other fuzzy rules with ant-based algorithms were examined in [23] and [25]. Notably, [35,36] was the first work to incorporate fuzzy decision making into the pick-up and drop probability calculations of an ant clustering algorithm and in [36] to show that by altering the pick-up and drop probability calculations, clustering improvements occur.

During the pick-up and drop process, an ant compares the object at its current location or the object it is holding to objects in nearby grid cells (the ant's neighborhood or field of view). An alteration to the size of the field of view directly affects the degree of spatial separation of objects within the grid. With a small field of view ants form clusters slowly and in a scattered manner. Unfavorably, this allows very different clusters to form adjacent to each other and, conversely, for small clusters containing similar items to form far apart. Previous work has addressed this trade-off with a variable field of view (also called a search radius) that grows during the clustering process [17, 37]. However, this progressive growth can lead to the destruction of small desirable clusters. This problem is exacerbated in datasets that possess a large variance in class populations.

In this paper, we also modify ant perception, but we do so in a fashion that does not require rule sets, categories, variable fields of view, or the incorporation of non-ant clustering algorithms. Instead, we use locally weighted regression kernel functions [1] as part of the ant's calculation of similarity of objects in its field of view. We also allow ants to hill-climb kernel function gradients in order to improve their chances of dropping objects in favorable locations. Compared to similarity functions in which objects are equally weighted, an appropriate kernel function can make the ants more sensitive to cluster shapes, sizes, and gradients, which leads to faster and higher quality clustering.

3 A Basic Ant Clustering Algorithm

A basic ant clustering algorithm is shown in Algorithm 1. This algorithm, which serves as a starting point for our contributions in this paper, is an adaptation of ATTA presented in [17]. Like [17] our ants always hold objects and teleport directly to objects when picking up. This basic algorithm does not include the ATTA enhancements

of a modulating neighborhood function, the increasing radius of perception and the modified pick-up and drop functions that this enhancement uses, or the adaptive α component. These are not included as the modulating neighborhood function that penalizes dissimilarities is an effect of the addition of the kernel functions which will be added later. Additionally, the increasing radius of perception and adaptive α component are not included to underscore the improvements that the kernel functions and similarity memory model provide.

Algorithm 1 Ant Clustering Algorithm

1: **// Initialization**
2: **for** each object $d_i \in D$ **do**
3: randomly place d_i on the grid
4: **for** each ant $a \in A$ **do**
5: **while** $\neg holding(a)$ **do**
6: randomly select an object d_i not being carried by any ant
7: $f(d_i) = \max\left(0, \frac{1}{s^2}\sum_{d_j \in L}\left[1 - \frac{\delta(d_i,d_j)}{\alpha}\right]\right)$
8: $p_{pick}(d_i) = \left(\frac{k^+}{k^+ + f(d_i)}\right)^2$
9: **if** $rand[0,1] \leq p_{pick}$ **then**
10: teleport ant a to location of d_i
11: $pickup(a, d_i)$
12: **// Main loop**
13: **for** each iteration t **do**
14: randomly select an ant $a \in A$
15: $d_i = carried\_item(a)$
16: $f(d_i) = \max\left(0, \frac{1}{s^2}\sum_{d_j \in L}\left[1 - \frac{\delta(d_i,d_j)}{\alpha}\right]\right)$
17: $p_{drop}(d_i) = \left(\frac{f(d_i)}{k^- + f(d_i)}\right)^2$
18: **if** $rand[0,1] \leq p_{drop}$ **then**
19: **// Drop current object and pick-up a new one**
20: $drop(a)$
21: **while** $\neg holding(a)$ **do**
22: randomly select an object d_i not being carried by any ant
23: $f(d_i) = \max\left(0, \frac{1}{s^2}\sum_{d_j \in L}\left[1 - \frac{\delta(d_i,d_j)}{\alpha}\right]\right)$
24: $p_{pick}(d_i) = \left(\frac{k^+}{k^+ + f(d_i)}\right)^2$
25: **if** $rand[0,1] \leq p_{pick}$ **then**
26: teleport ant a to location of d_i
27: $pickup(a, d_i)$
28: **else**
29: randomly move a in the grid
30: **// Drop all items at current locations**
31: $\forall a \in A, drop(a)$

Clustering takes place in a two dimensional grid of equal width and height which the ants traverse as a torus (left-right and top-bottom wrap around). The algorithm begins with an initial random placement of objects on the grid, as seen in lines 2 and 3 of Algorithm 1.

Next, each ant picks up an object using the process shown in lines 4 – 11. The ant randomly selects an object, d_i, that is not being carried by another ant. The ant then computes a measure of similarity between d_i and its neighbor objects, $f(d_i)$, and a pick-up probability, $p_{pick}(d_i)$, based on the measure of similarity. The ant generates

a random number $p \in [0, 1]$. If $p < p_{pick}$, the ant teleports to the location of object d_i and picks up the object. Otherwise, the ant repeats the process.

The measure of similarity or local density, $f(d_i)$, is the summation of the difference between object d_i and all other objects in d_i's neighborhood, L, calculated as

$$f(d_i) = \max\left(0, \frac{1}{s^2} \sum_{d_j \in L} \left[1 - \frac{\delta(d_i, d_j)}{\alpha}\right]\right). \quad (1)$$

Object dissimilarity, $\delta(d_i, d_j)$, is the Euclidean distance of objects d_i and d_j in the object attribute space and is unique to each clustering problem. Object dissimilarity is scaled by the constant α, which is data dependent, and provides a threshold that separates similar objects from dissimilar objects. The α constant must be tuned for each dataset to provide the best separation between objects. In ATTA [18], a mechanism to identify an appropriate α is included. Our implementation excludes this and uses a single α value per dataset to demonstrate the performance of the kernel and memory modifications. The neighborhood consists of the grid cells residing in a square centered on object d_i's current location. Each side of the square is $2r + 1 = s$ cells long, where r is the radius of the ant's vision. The number of the cells in the neighborhood is $(2r + 1)^2 = s^2$.

The probability of picking up an object, $p_{pick}(d_i)$, given by

$$p_{pick}(d_i) = \left(\frac{k^+}{k^+ + f(d_i)}\right)^2, \quad (2)$$

where the parameter $k^+ \in [0, 1]$ controls the pick-up sensitivity. The higher the k^+ value, the more likely it is that similar objects will be picked up (objects have to be highly dissimilar to trigger a pick-up).

Once each ant is holding an object, the main loop of the algorithm begins (line 13). Each iteration of the main loop proceeds as follows. An ant is selected at random. Using its current position on the grid, the selected ant computes the similarity measure, $f(d_i)$ for the object it is holding, d_i, using Equation 1. The ant generates a random number $p \in [0, 1]$ and drops its object if $p < p_{drop}(d_i)$, where $p_{drop}(d_i)$ is calculated as

$$p_{drop}(d_i) = \left(\frac{f(d_i)}{k^- + f(d_i)}\right)^2. \quad (3)$$

If the cell is empty, the ant drops the object in its current cell. Otherwise, the ant drops the object in the nearest empty cell. The nearest cell is found by checking the edges of concentric squares surrounding the ant's current location in a clock-wise direction.

The drop probability proposed by [6], Equation 3, provides a smooth response similar to the probability of picking up an object. The drop probability, p_{drop}, is higher when the carried object is similar to those in the ant's neighborhood and lower when dissimilar. The constant k^- governs the drop sensitivity; as the value of k^- increases, then more similar objects in the neighborhood are required to trigger a drop.

If the ant dropped its object, the ant goes through the process of picking up another object (lines 21– 27). This pick-up process is identical to the one described previously for algorithm initialization. If the object was not dropped because $p \geq p_{drop}(d_i)$, then the ant randomly moves to another cell in the grid (line 29).

Once all time steps are complete or some other termination condition has been met, the main loop ends and all ants drop their objects at their current locations (line 31).

Thus ends the basic ant clustering algorithm that is the starting point for our enhancements in the next section.

4 AntKSiMM

This section describes the kernel function and memory model enhancements for the ant clustering algorithm discussed in Section 3 and shown in Figure 1. After explaining each enhancement individually, we present a modified ant clustering algorithm called AntKSiMM that includes kernel functions, and the memory model system. Additionally, the effect each of these enhancements has on the performance of ant clustering is discussed.

4.1 Locally Weighted Ant Perception - Kernel Functions

As already seen in Section 3, the ant clustering algorithm weighs each object in an ant's neighborhood equally when doing similarity calculations. In this paper, we introduce a kernel function, $K(x)$, to the local density expression of Equation 1 to get

$$f(d_i) = \max\left(0, \frac{1}{S}\sum_{d_j \in L} K(x)\left[1 - \frac{\delta(d_i, d_j)}{\alpha}\right]\right). \tag{4}$$

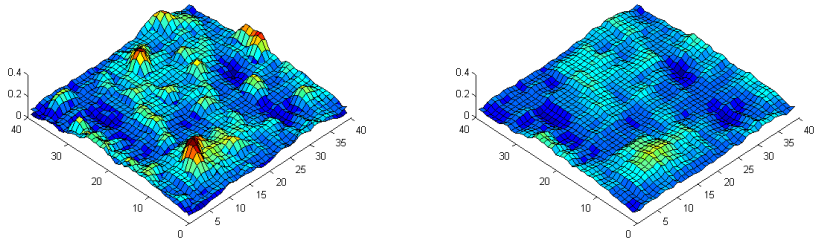
The x of the kernel function, $K(x)$, is the Euclidean distance in grid-space between objects d_i and d_j accounting for a wrap-around grid: $x = E(d_i, d_j)$. Note that object d_i is always assumed to be at the center of the neighborhood, L, being examined. Kernel functions allow for a non-uniform weighting of the objects in the neighborhood with respect to the object the ant is holding or is considering picking up, d_i. Kernel functions can be pre-computed and applied as masks in order to increase computation speed.

In order to normalize object similarities onto the interval $[0, 1]$, the summation in Equation 4 is scaled by S, which is the sum of every kernel weight within the ants' vision radius, r (Equation 5). Since r is constant, S can be pre-computed as

$$S = \sum_{a=-r}^{r}\sum_{b=-r}^{r} K(\sqrt{a^2 + b^2}). \tag{5}$$

Because they provide an alternate view of the organization of objects in the grid, kernels can more accurately direct ants to cluster centers. Section 4.3 explains how ants in the new algorithm will make use of (hill-climb) kernel gradients as they search for suitable drop locations.

A visual presentation of the information the kernel functions provide about the clustering space is shown in Figure 1. Figure 1 shows surface plots of the perceived density at each location on the grid for clusters of Iris data where Iris Setosa is similar and Iris Versicolor and Iris Virginica are dissimilar. Both plots are from the same data distribution and use the same vision radius. The surface shown in Figure 1(a) was



(a) (b)

Fig. 1 Perceived Density: (a) Gaussian kernel, and (b) Uniform kernel.

Table 1 Kernel functions, $K(x)$, where x is the Euclidean distance in grid-space between objects d_i and d_j, $x = E(d_i, d_j)$.

$Uniform(x) = 1$	$Gaussian(x) = e^{-\frac{x^2}{r^2}}$
$Inverse(x) = \frac{1}{x}$	$Triangular(x) = 1 - \frac{x}{r}$
$InverseDist(x) = \frac{1}{1+x}$	$Quadratic(x) = \max\left(0, \left(1 - \frac{x^2}{r^2}\right)\right)$
$InverseSquare(x) = \frac{1}{x^2}$	$Tricube(x) = max\left(0, \left(1 - \frac{x^3}{r^3}\right)\right)$
$Exponential(x) = e^{-x}$	

made using a Gaussian kernel and has a diversity of densities (high peaks) with very few plateaus. In contrast, Figure 1(b) used a uniform kernel and shows some maxima and minima, but none as dramatic as ones produced by the Gaussian kernel. The relatively flat landscape of the uniform kernel prevents ants from focusing their clustering efforts on areas of high similarity that are more clearly revealed by the Gaussian kernel.

In this paper, we investigate the effects of nine kernel functions common to locally weighted regression as shown in Table 1 [1]. Figure 2 shows the weighting effect each kernel function has on similarity weights when $r = 4$. Note that the basic ant clustering algorithm of Algorithm 1 uses the uniform kernel function since $K(x)$ is equal to 1 for all objects and $S = s^2$.

4.2 The Similarity Memory Model (SiMM)

As reviewed in the related work section, several past ant clustering algorithms have experimented with limited ant memories [17, 24, 29]. While these memory systems improved clustering performance, we note that any memory replacement scheme that is based upon the frequency of use (such as FIFO, least recently used, etc.) creates a bias in the memory that results in a loss of information about low-frequency samples. To overcome this bias one could employ a scheme that replaces objects with the highest error rate, such as the neural network approach of [32]. However, in clustering there is no mechanism to calculate error as in classification. A better way to achieve memory diversity (cover the sample space) is to update the objects in memory based on object

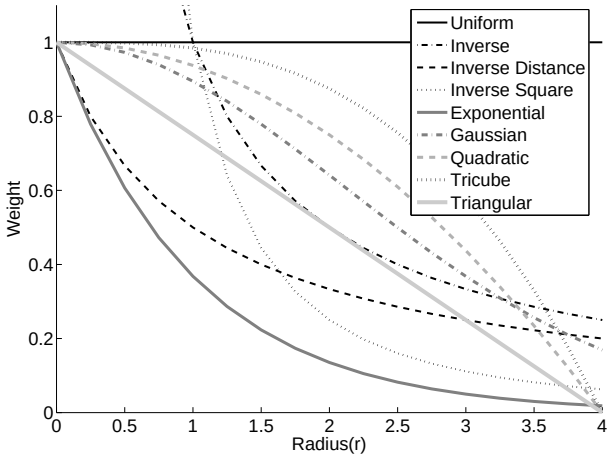


Fig. 2 Kernel Function Plots

similarity. By updating the most similar objects in memory, the ants are more likely to have memory objects from each of the clusters.

To this end, we propose the Similarity Memory Model (SiMM) for ant clustering. Under SiMM, an ant has some probability of using its memory to determine a favorable location to which to move. The decision is governed by a temperature variable $\tau_i \in [0, 1]$ unique to each object, i. The ant compares its held object's temperature, τ_i, to a random value between 0 and 1. If the random value is less than τ_i, then the ant jumps to the grid cell where the most similar memory-stored object was dropped. The ant is said to have made a *memory jump*. After the memory jump, the ant "cools" the temperature of the object it is holding by reducing the value of τ_i by a small amount, k_τ. If τ_i becomes less than the temperature threshold, T, then τ_i is set to zero. This cooling suggests a *deterministic annealing* procedure for clustering in which the exploitative nature of the ants is determined through a sequence of phase transitions by continuously decreasing τ_i following an *annealing schedule* [34]. The deterministic annealing process is included in all tests using the same parameter settings.

Upon dropping an object, the dropped object replaces the memory entry that is most similar to the dropped object. This occurs regardless of how the ant moved in the grid prior to its decision to drop. If the memory is not full, the dropped object is simply added to the memory.

In this paper, we compare the performance of the Similarity Memory Model to a first in first out (queue) replacement scheme [24], and a weighted random replacement scheme. In the random scheme, the ant randomly selects and replaces a memory object with the dropped object according to a weighting of memory objects based on their similarity to the dropped object (highly similar objects are more likely to be replaced).

In order to stay true to the simple nature of ants and to not overshadow the stochastic nature of the clustering process itself, we employ a small 20-item memory in our experiments. Note that the memory is orders of magnitude smaller than the number of data items.

Evidence that SiMM clusters better than a queuing memory replacement scheme is shown in Figure 3. Figure 3a is the result of clustering on the Lincoln Lab Intrusion Detection dataset when using SiMM. Clustering on the same dataset using a queueing

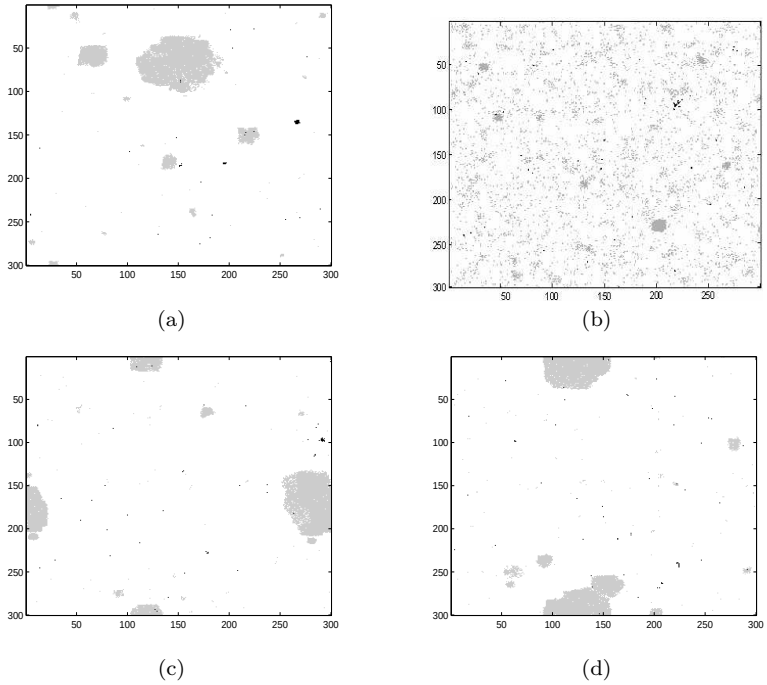


Fig. 3 The Power of the Similarity Memory Model. Lincoln Lab Intrusion Data clustered using the (a) Similarity Memory Model, (b) no memory, (c) the weighted random approach, and (d) a queue-based scheme. All data collected at step 50,000, using the uniform kernel, and the settings in Table 2.

approach is shown in Figure 3d. Unlike the queued replacement scheme, SiMM is able to cluster the intrusions (black samples) into fewer and more distinct groups and forms larger clusters of normal traffic (gray samples). Note that intrusions are sometimes found near normal traffic in both graphs. The weighted random scheme results in Figure 3c show that using similarity replacement even in a Monte Carlo way improves the results, as the network intrusions have some grouping, although not as much as with SiMM. Also, of note is that on this dataset, using any of the ant memory schemes has a positive effect on clustering as can be seen by comparing Figure 3b (no memory) to the other plots in Figure 3.

All three memory replacement schemes, SiMM, Queue, and Random, are implemented both as individual ant memories and as shared colony (global) memories. In a shared memory, all ants use one memory array. Thus, Unified-SiMM, Unified-Queue, and Unified-Random memory systems are the global memory versions of SiMM, FIFO, and Random.

4.3 AntKSiMM Algorithm

The AntKSiMM algorithm shown in Algorithm 2 is the basic ant clustering algorithm of Algorithm 1 with our kernel and memory system extensions. Because of this, our explanation of AntKSiMM focuses on the changes.

The drop and pick-up procedures differ from the basic algorithm in that the local density calculations of lines 7, 16, and 29 (see Equation 1), have been replaced by ones of the form of Equation 4 that use a kernel function calculated over the Euclidean distance between objects, $K(E(d_i, d_j))$, as discussed in Section 4.1.

When an object is dropped, the ant updates its memory in accordance with the SiMM described in Section 4.2 (lines 21-26). After the memory update, the ant then performs the pick-up process in lines 27 to 32.

Having selected a new object, the ant moves to that location and picks it up. The ant then randomly decides to perform a memory jump or make a random move by comparing a random number to the held object's temperature τ_i (line 34). If the ant jumped, the memory temperature of τ_i is cooled by a factor of $k_\tau \in (0, 1)$ as in line 37.

Another change to ant behavior, implemented in all version of the algorithm, is that when an ant decides not to drop an object, the ant makes a random move with probability k_g (line 40) or else hill-climbs the kernel gradient. When following the gradient, the ant computes the density, $f(d_i)$, for each cell adjacent to its current position and then moves to the cell with the highest density (lines 43–44).

At this point, the iteration is complete and the next iteration starts. At the completion of AntKSiMM, all ants drop their held items to ensure all objects are on the grid for proper evaluation of clustering.

5 Analyzing AntKSiMM Performance

AntKSiMM has been tested on the Iris and Wisconsin Breast Cancer data sets from the UCI Machine Learning Repository [30] and the Lincoln Lab Intrusion Detection System week 2 data [12]. The following subsections contain reports and analyses of AntKSiMM's performance, focusing on the impact of the Kernel functions and the Similarity Memory Models.

5.1 Performance Metrics

Three measures are used for comparison purposes: F-Measure, Adjusted Rand Index, and the number of clusters. A cluster is defined as a contiguous grouping of objects in the ant space. Contiguous means that two samples share an edge or a corner. Each metric is defined in more detail below.

5.1.1 F-Measure

The F-measure represents the harmonic mean between the precision and recall of the clustering for all classes. The precision, $p(i, j)$, for a single class, i, and cluster, j, is the proportion of items in that cluster that are of that class. The recall, $r(i, j)$, is the proportion of that class that belongs to that cluster. Equation 6 shows the derivation of an individual F-measure, where n represents the total number of objects being clustered, n_j is the number of objects within cluster j, n_i is the number of objects in class i, and n_{ij} is the number of objects of class i within cluster j.

Algorithm 2 AntKSiMM Algorithm

// Initialization
for each object $d_i \in D$ **do**
  randomly place d_i on the grid
for each ant $a \in A$ **do**
  while $\neg holding(a)$ **do**
    randomly select an object d_i not being carried by any ant
    $f(d_i) = \max\left(0, \frac{1}{S}\sum_{d_j \in L} K\left(E(d_i, d_j)\right)\left[1 - \frac{\delta(d_i,d_j)}{\alpha}\right]\right)$
    $p_{pick}(d_i) = \left(\frac{k^+}{k^+ + f(d_i)}\right)^2$
    if $rand[0,1] \leq p_{pick}$ **then**
      teleport ant a to location of d_i
      $pickup(a, d_i)$
// Main loop
for each iteration t **do**
  randomly select $a \in A$
  $d_i = carried\_item(a)$
  $f(d_i) = \max\left(0, \frac{1}{S}\sum_{d_j \in L} K\left(E(d_i, d_j)\right)\left[1 - \frac{\delta(d_i,d_j)}{\alpha}\right]\right)$
  $p_{drop}(d_i) = \left(\frac{f(d_i)}{k^- + f(d_i)}\right)^2$
  if $rand[0,1] \leq p_{drop}$ **then**
    // Drop current object and pick-up a new one
    $drop(a)$
    // Update memory M
    if memory is full **then**
      $best = \min_{m \in M} \delta(d_i, d_m)$
      replace $best$ with d_i
    else
      insert d_i into a free memory slot
    while $\neg holding(a)$ **do**
      randomly select an object d_i not being carried by any ant
      $f(d_i) = \max\left(0, \frac{1}{S}\sum_{d_j \in L} K\left(E(d_i, d_j)\right)\left[1 - \frac{\delta(d_i,d_j)}{\alpha}\right]\right)$
      $p_{pick}(d_i) = \left(\frac{k^+}{k^+ + f(d_i)}\right)^2$
      if $rand[0,1] \leq p_{pick}$ **then**
        $pickup(a, d_i)$
    // Check memory for jump
    if $rand[0,1] < \tau_i$ **then**
      $best = \min_{m \in M} \delta(d_i, d_m)$
      teleport ant to location of $best$
      $\tau_i = \tau_i \cdot k_\tau$
  else
    if $rand[0,1] < k_g$ **then**
      randomly move a in the grid
    else
      // Move a **according to gradient**
      Compute $f(d_i)$ for each grid cell adjacent to a's current position.
      Move a to adjacent cell which maximizes $f(d_i)$.
// Drop all items at current locations
$\forall a \in A, drop(a)$

$$\begin{aligned} p(i,j) &= \frac{n_{ij}}{n_j} \\ r(i,j) &= \frac{n_{ij}}{n_i} \\ F(i,j) &= \frac{2 \times p(i,j) \times r(i,j)}{p(i,j)+r(i,j)} \end{aligned} \tag{6}$$

The combined F-measure of Equation 7 is derived from the maximum F-measure that each class sees over all clusters, and is the mean of all class maximum F-measures weighted by class population. The F-measure is bound by the interval (0, 1], with higher values indicating a better clustering [14, 15].

$$F_{combined} = \sum_i \frac{n_i}{n} \max_j F(i,j) \tag{7}$$

5.1.2 Adjusted Rand Index

The Rand Index [33] compares the clustering output to the class membership, and represents the classification performance as a fraction of all object-cluster pairs that have been correctly clustered over all that have been incorrectly clustered. A problem with Rand Index and other external indices including F-measure is that it does not take a constant value when receiving two random partitions. This problem was solved with the Adjusted Rand Index [20], which does take zero if the index equals its expected value. Yeung and Ruzzo [38] show that the Adjusted Rand Index, which has a range of $[0, 1]$, typically distributes values further apart and therefore has a higher sensitivity compared to the Rand index. Letting c_{ij} be the count of objects of class i that reside in cluster j, and c being the number of objects, the Adjusted Rand Index is defined as:

$$\frac{\sum_{i,j} \binom{c_{ij}}{2} - \left[\sum_i \binom{c_{i\cdot}}{2} \sum_j \binom{c_{\cdot j}}{2}\right] / \binom{c}{2}}{\frac{1}{2}\left[\sum_i \binom{c_{i\cdot}}{2} + \sum_j \binom{c_{\cdot j}}{2}\right] - \left[\sum_i \binom{c_{i\cdot}}{2} \sum_j \binom{c_{\cdot j}}{2}\right] / \binom{c}{2}} \tag{8}$$

5.1.3 Number of Clusters

Clustering quality can be determined by how many clusters are formed; it is best if the clustering algorithm produces as many clusters as there are classes in the domain.

5.2 Testing Domains

AntKSiMM was tested using three classic datasets: Iris, Wisconsin Breast Cancer, and the Lincoln Lab Intrusion Detection System (IDS) data.

5.2.1 Iris

The Iris data [10, 30] contains 150 samples with four attributes concerning the sample plant's physical dimensions. Each sample is from one of three species of iris, where each species is represented by 50 samples. With three equally sized classes, by placing all objects of Iris into a single cluster an F-measure of 0.5 and adjusted Rand Index of 0.66 can be achieved. Previously, an ant clustering algorithm achieved an F-measure of 0.82 [16].

Table 2 Week 2 Attack Profile.

Day	Attack	Attack Type	Start Time	Duration
1	Back	DOS	9:39:16	00:59
2	Portsweep	Probe	8:44:17	26:56
3	SATAN	Probe	12:02:13	2:29
4	Portsweep	Probe	10:50:11	17:29
5	Neptune	DOS	11:20:15	4:00

In the Iris dataset, two of the three classes are linearly separable, with only a small amount of overlap occuring from the third class. Because of the straightforward nature of this dataset, the performance of AntKSiMM on artificial datasets should exhibit similar clustering as determined using the Iris dataset.

5.2.2 Wisconsin Breast Cancer

The Wisconsin Breast Cancer data [27, 30] contains 683 samples, 239 of which are positive for breast cancer while the rest are not. The dataset consists of nine attributes that are scaled linearly onto a hypercube [8]. The dataset also includes samples that contain missing data. In our testing, these samples were omitted rather than replacing the missing data and potentially modifying the sampling.

5.2.3 Lincoln Lab Intrusion Detection System (IDS)

This dataset comes from the Lincoln Laboratory of the Massachusetts Institute of Technology [12]. In this paper, we use the 1999 week 2 data, which is expert-identified normal traffic mixed with attacks. We follow the same data preparation methodology as [5], reducing the dimensionality of the data to three attributes: number of bytes per second, number of packets per second, and number of ICMP packets per second.

There are 5,147 benign samples and just 55 malicious, making for a very skewed clustering problem; a mere 1.06% of the samples have to be separated from the massive group of benign data. Table 2 describes the five kinds of malicious attacks in the dataset. The network traffic samples are generally categorized as either malicious or benign, with the five attacks as malicious, and all other traffic as benign.

5.3 Algorithm Settings

AntKSiMM was executed using the parameters and conditions shown in Table 3, which are largely based on previously published results [14]. Similarly, the grid was a perfect square that was at least ten times as many grid cells as data samples. The value of α is data dependent and was determined experimentally by repeatedly executing AntKSiMM with the basic uniform kernel function to search for an optimum value.

For each kernel function and memory strategy pairing, AntKSiMM is executed 30 times per dataset and the clustering results are averaged for each of the metrics used. The average performance is created by calculating the mean of each metric at every 500 iterations for the Iris and Wisconsin datasets and every 5,000 iterations for the IDS dataset from each of the 30 executions. Note that only one ant moves per iteration.

Table 3 Parameter and condition values.

Description	Symbol	Value
Neighborhood Radius	R	4
Number of Memory Entries		20
Initial Temperature of Memory Objects	τ_i	1.0
Memory Cooldown Multiplier	k_τ	0.9
Memory Temperature Threshhold	T	0.9^{10}
Pick-up Sensitivity Constant	k^+	0.30
Drop Sensitivity Constant	k^-	0.30
Hill-Climbing Decision Constant	k_g	0.7
Dissimilarity Scaling Constant (Iris)	α	0.25
Dissimilarity Scaling Constant (Cancer)	α	0.39
Dissimilarity Scaling Constant (Intrusion)	α	0.01

5.4 Kernel Function Performance

To illustrate the advantage of adding kernel functions to the basic clustering algorithm, this section compares the performance of the nine kernel functions (Table 1) on all three datasets using the Queue memory model.

5.4.1 Kernel Functions on the Iris Dataset

Iris results are shown in Figure 4. Figures 4a and 4b display F-measure and Adjusted Rand Index performance as functions of time, thus giving an indication of how quickly each kernel clusters. Note that all kernels except for Exponential work more quickly than the Uniform kernel (the one used by the default ant clustering algorithms).

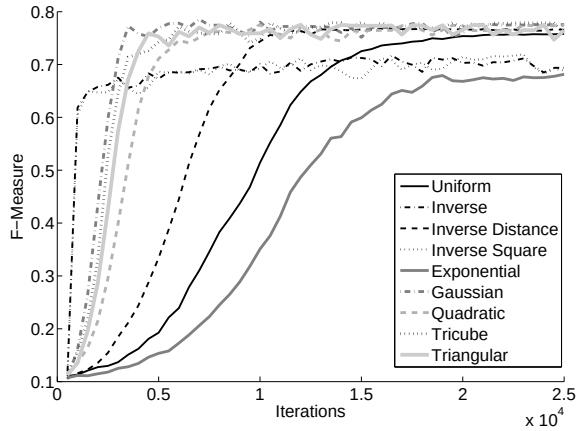
An examination of Figure 4c reveals that all kernels except Exponential (again) result in fewer clusters than the Uniform kernel and, therefore, more closely match the Iris dataset's three classes of species. As Table 4 shows, the Gaussian, Quadratic, Tricube and Triangular average just over three clusters each, followed by the Inverse Distance kernel with a cluster count average of 7.2. The Uniform, Inverse, and Inverse Square then perform the worst, with cluster counts in the 10 to 12 clusters range.

Kernels can be put into four performance-based groups. These groupings also occur in the Wisconsin Breast Cancer and Lincoln Lab Intrusion datasets and are clearly visible in Table 4. In descending order of performance they are:

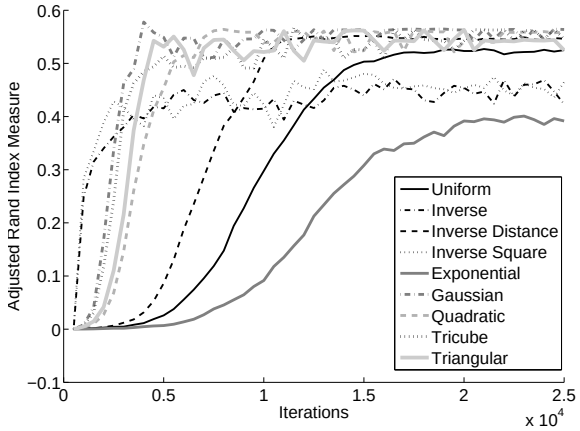
1. Gaussian, Quadratic, Tricube, and Triangular. These functions, which all happen to be smoothly decaying, have the highest clustering performance in terms of F-Measure and Adjusted Rand Index scores. Although they cluster rapidly, they are not the fastest.
2. Inverse, Inverse Distance, and Inverse Square. These kernels are the quickest to cluster, but suffer from poor overall clustering quality since they weight nearby objects more heavily than distant objects. This tends to create loosely grouped clusters.
3. Uniform. Converges slowly, but solutions rival the kernels in group #1.
4. Exponential. Has the worst cluster quality and speed.

5.4.2 Kernel Functions on the Wisconsin Breast Cancer Dataset

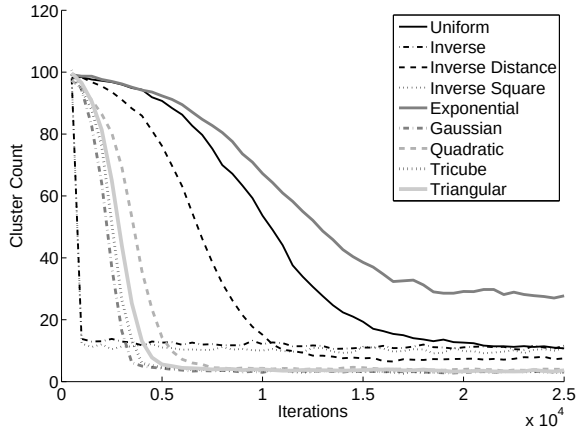
Similar to the Iris dataset, the smoothly decaying functions (Gaussian, Quadratic, Tricube, and Triangular) result in better clusters as shown in Table 4 and the F-Measure



(a)



(b)



(c)

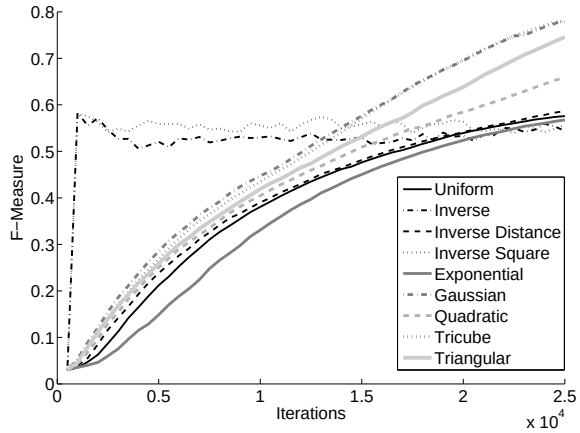
Fig. 4 Kernel performance of over time on the Iris dataset: (a) F-Measure, (b) Adjusted Rand Index and (c) cluster count.

Table 4 End-of-run means and standard deviations for the kernel function comparison tests.

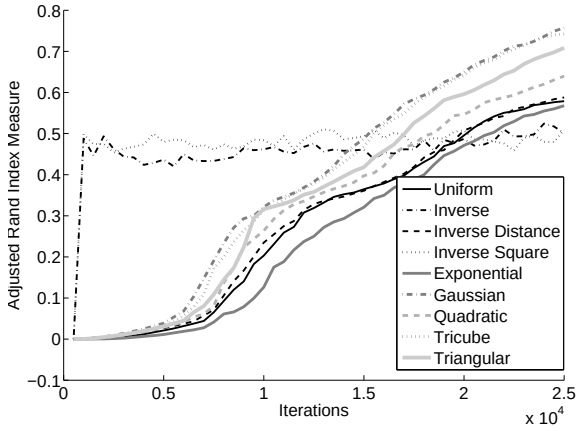
	Iris		Cancer		Intrusion	
Kernel	Mean	StdDev	Mean	StdDev	Mean	StdDev
Cluster Count						
Uniform	10.93	2.26	155.00	48.07	485.70	74.58
Inverse	12.63	3.49	26.77	5.98	773.77	86.05
Inv Dist	7.90	2.40	102.60	32.21	315.37	44.63
Inv Square	9.87	3.89	28.97	6.92	229.57	28.27
Exponential	31.23	4.72	169.87	14.63	535.67	70.82
Gaussian	3.17	1.10	16.13	2.88	106.93	10.23
Quadratic	3.67	0.94	33.17	11.33	161.67	15.82
Tricube	3.27	0.93	18.57	5.43	117.77	10.98
Triangular	3.57	1.15	23.03	7.86	139.63	10.92
F-measure						
Uniform	0.76	0.01	0.69	0.10	0.88	0.06
Inverse	0.68	0.07	0.57	0.08	0.57	0.14
Inv Dist	0.76	0.01	0.77	0.09	0.86	0.06
Inv Square	0.71	0.07	0.54	0.08	0.56	0.18
Exponential	0.64	0.05	0.65	0.01	0.88	0.06
Gaussian	0.77	0.05	0.87	0.09	0.86	0.06
Quadratic	0.77	0.05	0.86	0.09	0.90	0.06
Tricube	0.78	0.00	0.89	0.09	0.89	0.05
Triangular	0.77	0.05	0.92	0.05	0.87	0.06
Adjusted Rand Index						
Uniform	0.52	0.02	0.75	0.13	0.80	0.10
Inverse	0.40	0.10	0.54	0.11	0.36	0.14
Inv Dist	0.54	0.02	0.80	0.05	0.77	0.10
Inv Square	0.46	0.10	0.48	0.11	0.37	0.18
Exponential	0.33	0.06	0.76	0.01	0.79	0.09
Gaussian	0.55	0.10	0.86	0.07	0.76	0.09
Quadratic	0.54	0.10	0.85	0.07	0.83	0.09
Tricube	0.56	0.01	0.87	0.08	0.81	0.08
Triangular	0.54	0.10	0.89	0.04	0.79	0.09

and Adjusted Rand Index graphs of Figures 5a and 5b, respectively. Additionally, half as many clusters are created by these kernels as with the Uniform kernel, Figure 5c. While better, no kernel came close to producing the ideal of two clusters, although the smootly decaying functions performed better.

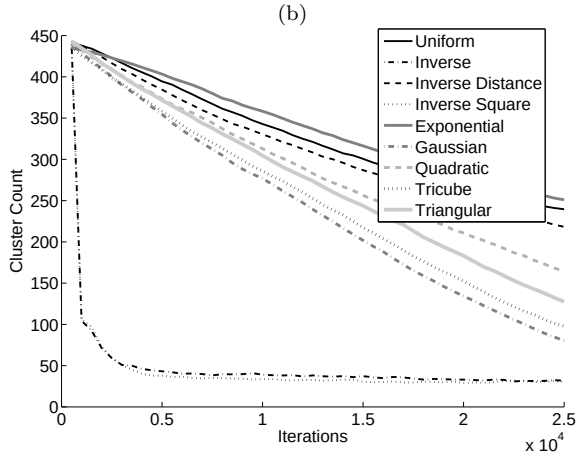
Unlike the Iris dataset, the variation in clustering speed is not as pronounced here. The exceptions are the inverse kernels (Inverse, Inverse Distance, and Inverse Square) which cluster quickly but, as a class, have lower F-measure and Adjusted Rand Index scores than the Gaussian, Quadratic, Tricube, and Triangular kernel functions. We suspect that poorer clustering speed on the Wisconsin dataset has to do with the noise and the non-linearly separable nature of the classes. The noise occurs because of the missing data and incorrect data (survey respondents misrepresented information) for some of the samples. The Wisconsin dataset also presents problems for classification algorithms as well [27, 30], which makes it that much more of a difficult dataset for clustering. However, we are encouraged by the fact that the F-Measure and Adjusted Rand Index curves improve over time.



(a)

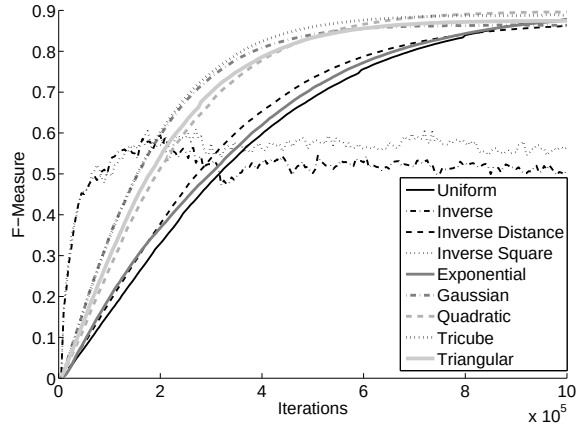


(b)

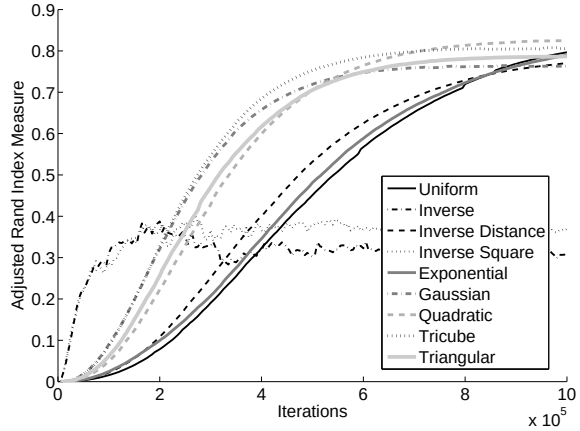


(c)

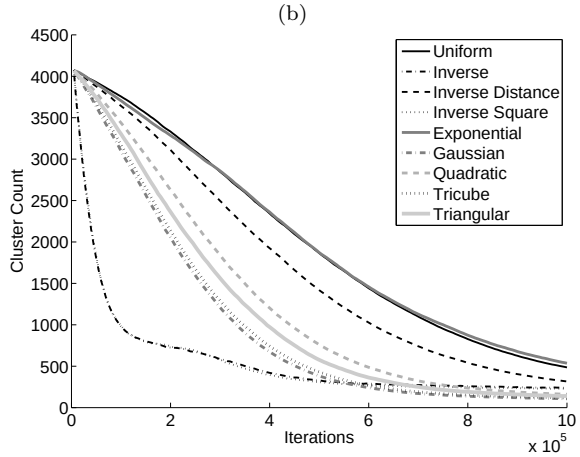
Fig. 5 Kernel performance over time on the Wisconsin Breast Cancer dataset: (a) F-Measure, (b) Adjusted Rand Index and (c) cluster count.



(a)



(b)



(c)

Fig. 6 Kernel performance of over time on the Lincoln Lab Intrusion Dataset (IDS): (a) F-Measure, (b) Adjusted Rand Index, and (c) cluster count.

5.4.3 Kernel Functions on the Lincoln Lab Intrusion Dataset

Table 4 and Figure 6 show the F-measure, Adjusted Rand Index, and cluster count trends for the Lincoln Lab Intrusion Dataset (IDS). The F-Measure and Adjusted Rand Index show similar trends, and results reveal that the smoothly decaying functions (Gaussian, Quadratic, Tricube, and Triangular) provide the best compromise between clustering speed and clustering performance compared to the other kernel functions. This observation continues the trend seen with the Iris and Wisconsin datasets.

The smoothly decaying functions continue to outperform the other kernel functions in terms of cluster count as shown in Figure 6b and Table 4. The smoothly decaying kernels result in 1/4 as many clusters as does the Uniform kernel. The cluster count is important in this domain, because if the system is to respond to live network traffic, the more clusters that must be checked for membership, the longer the processing delay will be.

One problem to underscore is that although the kernel functions reduce the cluster count, there are still many more clusters than there are classes in the Intrusion dataset. At best, the cluster count is only reduced to 110 with one of the clusters representing 85% of all objects. Fortunately, the similarity based memory-model improves further on this as presented in the following section.

5.5 Similarity Memory Model Results

This section compares the results from each of the six memory strategies: Queue, Random, Similarity (SiMM), Unified Queue, Unified Random, and Unified Similarity. Recall that the Unified strategies employ a single colony memory rather than each ant having its own local memory. The comparison of the six memory strategies is conducted on the Iris and Lincoln Lab Intrusion datasets using the Gaussian kernel function which was well-rounded performer on the kernel tests. The Wisconsin Breast Cancer dataset is omitted from the memory results because there was no difference in the memory strategies due to the noise present in the dataset.

5.5.1 Memory Strategies on the Iris Dataset

On the Iris dataset, the Similarity strategy is the clear winner. In Figure 7a, the Adjusted Rand Index results curve shows that the clustering under the Similarity strategy passes an index of 0.5 in less than 2,000 iterations, progressing almost twice as fast as any other strategy tested. Although difficult to see in Figure 7b, the Similarity strategy reaches a 30-run average of three clusters at about 4,000 iterations. End-of-run numbers shown in Table 5 show a rough equivalence between all memory strategies in terms cluster counts, F-measure, and Adjusted Rand Index. Overall, the results show that a small memory of the correct type can cut the clustering time in half and bring the cluster count in-line with the number of classes in the dataset.

Additionally, as originally identified by Montes de Oca et al. [29], a colony wide memory (the Unified models) offers no improvements over the corresponding individual strategy. Our results also reveal that there is no advantage to be gained from the Unified strategies. Indeed, in the case of SiMM and Unified-SiMM, the unified version actually clusters more slowly. The most likely reason for the lackluster performance of the unified memories is that they contain a subsampling of the cluster space in a single

Table 5 End-of-run means and standard deviations for the memory comparison tests.

Memory	Iris		Intrusion	
Type	Mean	StdDev	Mean	StdDev
Cluster Count				
Queue	3.17	1.10	106.93	10.23
Random	3.07	0.96	105.60	7.08
SiMM	2.47	0.56	79.07	7.04
Unified-Queue	3.20	0.98	106.53	11.52
Unified-Random	2.77	0.80	106.10	8.31
Unified-SiMM	3.17	0.82	132.50	11.07
F-measure				
Queue	0.77	0.05	0.86	0.06
Random	0.76	0.07	0.87	0.06
SiMM	0.75	0.09	0.87	0.06
Unified-Queue	0.77	0.05	0.87	0.06
Unified-Random	0.77	0.05	0.88	0.06
Unified-SiMM	0.78	0.01	0.52	0.16
Adjusted Rand Index				
Queue	0.55	0.10	0.76	0.09
Random	0.53	0.14	0.77	0.10
SiMM	0.52	0.17	0.77	0.09
Unified-Queue	0.55	0.10	0.78	0.09
Unified-Random	0.55	0.10	0.79	0.09
Unified-SiMM	0.56	0.01	0.32	0.15

memory object and thus fail to capture much of the diversity. By having individual ants maintain their own memories, each ant potentially becomes an expert in a particular experience subspace.

5.5.2 Memory Strategies on the Lincoln Lab Intrusion Dataset

In clustering the much larger Lincoln Lab Intrusion dataset, the Similarity memory model provides for a slightly faster clustering than the other memory strategies (Figure 8a). The F-measure and Adjusted Rand Index scores for the better performing memory schemes are nearly tied with no statistical significance in their performance. However, Similarity produced 22.1% fewer clusters than Unified Random as shown in Figure 8b and Table 5. Additionally, as with the Iris dataset, the Unified Similarity memory model is the worst performing memory model. Again this is likely due to the overrestrictive subsampling of the clustering space that the model uses.

None of the clusterings produced came close to grouping the data into the two sets of clean and anomalous network traffic. This underscores the skewed nature of this dataset and shows that ant clustering in the domain is difficult.

6 Conclusion

The AntKSiMM results demonstrate a doubling of clustering speed and a significant reduction in the cluster count for real world datasets over a basic ant clustering algorithm. These improvements are the results of two contributions presented in the paper. The first enhancement is the addition of kernel functions to the density function calculation. The kernel functions provide the ants with a steeper decision gradient so that they may more easily locate desirable drop locations. Results show that smoothly

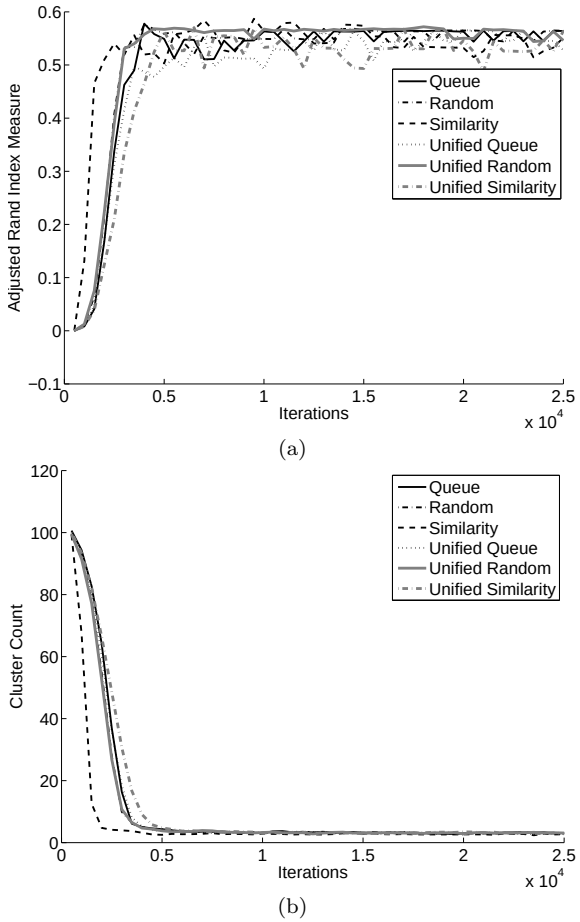


Fig. 7 Effect of memory strategies on clustering for the Iris dataset: (a) Adjusted Rand Index and (b) cluster count.

decaying kernel functions result in the best overall performance gains with respect to both speed and clustering quality.

The second contribution in AntKSiMM is the Similarity Memory Model (SiMM). This memory model was developed to improve clustering on large and very skewed datasets like the Lincoln Lab Intrusion dataset, where there are only a few items of one class. Without a memory at all, clustering is close to impossible as seen in Figure 3. However, under SiMM, a dominant cluster forms, and the attacks are clustered into a few groups. For both the Iris and Intrusion datasets, SiMM achieves a smaller cluster count while also speeding clustering. These benefits permit the ant clustering paradigm to be applied to larger datasets than previously attempted.

As noted in the algorithm discussion, the radius increasing enhancement found in ATTA is not included in our algorithm, but is a candidate for future investigation. Future work also includes a direct comparison and incorporation of the kernel functions and SiMM into ATTA and other ant clustering algorithms. Also, since the ant clustering algorithm is performing an attribute space reduction from n attributes to two spatial

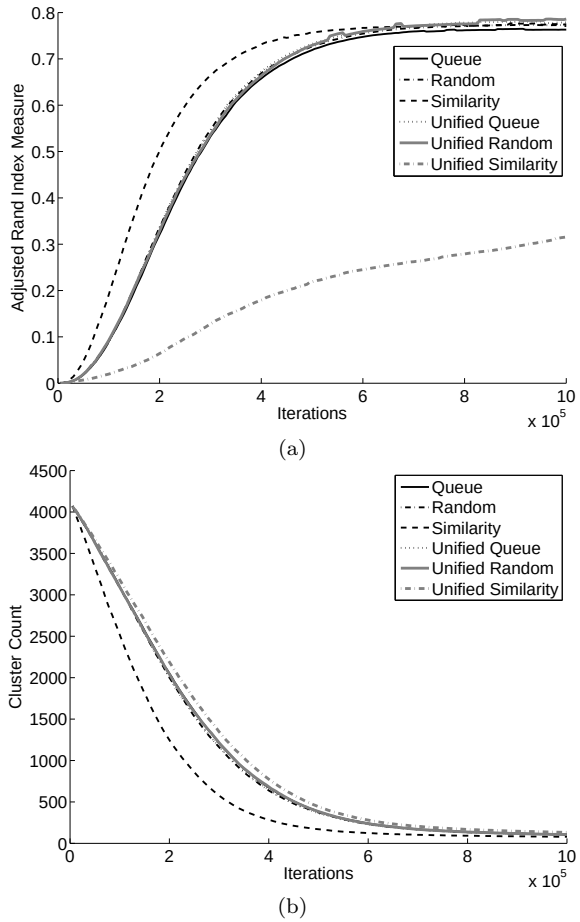


Fig. 8 Effect of memory strategies on clustering for the Intrusion dataset : (a) Adjusted Rand Index and (b) cluster count.

attributes, an open question is if this reduction is the best reduction for all datasets. As has been shown in Sammon [22], a correct nonlinear mapping can greatly assist in classification. An analysis along these lines could not only include the nonlinear mapping, but also feature selection, feature saliency, and feature reduction and/or combination techniques to improve the clustering performance of the ant clustering algorithm.

References

1. C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.
2. P. Cheeseman and J. Stutz. Bayesian classification (autoclass): theory and results. In *Advances in knowledge discovery and data mining*, pages 153–180. AAAI Press, Menlo Park, CA, 1996.
3. L. Chen, Y. Liue, C. Fattah, and G. Yan. HDACC: A Heuristic Density-Based Ant Colony Clustering Algorithm. In *IEEE/WIC/ACM International Conference on Intelligent Agent*

Technology (IAT 2004), pages 397–400. IEEE Computer Society Press, Los Alamitos, CA, 2004.
4. R. Cucchiara. Analysis and comparison of different genetic models for the clustering problem in image analysis. In *International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 423–427. Springer-Verlag, Berlin, Germany, 1993.
5. D. Dasgupta and F. Gonzales. An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, 6:179–188, 2002.
6. J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting robot-like ants and ant-like robots. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 356–363. MIT Press, Cambridge, MA, 1990.
7. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.
8. T. Ekola, M. Laurikkala, T. Lehto, and H. Koivisto. Network traffic analysis using clustering ants. In *Proceedings of the 17th World Automation Congress*, pages 275–280. IEEE Computer Society Press, Los Alamitos, CA, 2004.
9. D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.
10. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7(Part II):179–188, 1936.
11. J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1990.
12. J. Haines, R. Lippmann, D. Fried, E. Tran, S. Boswell, and M. Zissman. DARPA intrusion detection system evaluation: Design and procedures. Technical report, MIT Lincoln Laboratory Technical Report, 1999.
13. G. Hamerly and C. Elkan. Learning the k in k-means. In *Advances in Neural Information Processing Systems*, pages 281–288. MIT Press, Cambridge, MA, 2003.
14. J. Handl. Ant-based methods for tasks of clustering and topographic mapping: Extensions, analysis and comparison with alternative methods. Master's thesis, Universität Erlangen-Nürnberg, Germany, 2003.
15. J. Handl, J. Knowles, and M. Dorigo. Ant-based clustering: a comparative study of its relative performance with respect to k-means, average link, and 1d-som. Technical Report TR/IRIDIA/2003-24, Université Libre de Bruxelles, 2003.
16. J. Handl, J. Knowles, and M. Dorigo. On the performance of ant-based clustering. *Frontiers in Artificial Intelligence and Applications*, 104:204–213, 2003.
17. J. Handl, J. Knowles, and M. Dorigo. Ant-based clustering and topographic mapping. *Artificial Life*, 12(1):35–61, 2006.
18. J. Handl, K. Knowles, and D. Kell. Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21(15):3201–3212, 2005.
19. J. Handl and B. Meyer. Ant-based and swarm-based clustering. *Swarm Intelligence*, 1(2): 95–113, 2007.
20. L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.
21. D. R. Jones and M. A. Beltrano. Solving partioning problems with genetic algorithms. In *Fourth International Conference on Genetic Algorithms*, pages 442–449. Morgan Kaufmann, Burlington, MA, 1991.
22. J.W. Sammon Jr. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, 1969.
23. P.M. Kanade and L.O. Hall. Fuzzy ant clustering by centroid positioning. In *Proceedings of the 2004 IEEE International Conference on Fuzzy Systems*, pages 371–376. IEEE Computer Society Press, Los Alamitos, CA, 2004.
24. E. B. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 501–508. MIT Press/Bradford Books, Cambridge, MA, 1994.
25. P. Lučić. *Modelling Transportation Systems Using Concepts of Swarm Intelligence and Soft Computing*. PhD thesis, Virginia Polytechnic Institute, 2002.
26. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, San Francisco, CA, 1967.
27. O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18, 1990.

28. N. Monmarché. On data clustering with artificial ants. In A. A. Freitas, editor, *Data Mining with Evolutionary Algorithms: Research Directions – AAAI-99 and GECCO-99 Workshop*, pages 23–26. AAAI Press, Menlo Park, CA, 1999.
29. M. A. Montes de Oca, L. Garrido, and J. L. Aguirre. A first approach to study the effects of direct information exchange between agents in ant-based clustering. In S. Kumar, A. Abraham, J. Harnisch, and A. Satyadas, editors, *Proceedings of the First World Congress on Lateral Computing WCLC 2004*. World Federation on Lateral-Computing, Bangalore, India, 2004.
30. D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998.
31. D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Fifth ACM SIGKDD International Conference OnKnowledge Discovery and Data Mining (KDD-99)*, pages 277–281. ACM, New York, NY, 1999.
32. D. Pomerlau. Knowledge-based training of artificial neural networks for autonomous robot driving. In J.H. Connel and S. Mahadevan, editors, *Robot Learning*. Kluwer Academic Publishers, 1993.
33. W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.
34. K. Rose, E. Gurewitz, and G. C. Fox. Statistical mechanics and phase transitions in clustering. *Physical Review Letters*, 65(8):945–948, 1990.
35. S. Schockaert, M. De Cock, C. Cornelis, and E. E. Kerre. Efficient clustering with fuzzy ants. In *Applied Computational Intelligence, Proceedings of the 6th International FLINS Conference*, pages 195–200. World Scientific, Singapore, 2004.
36. S. Schockaert, M. De Cock, C. Cornelis, and E. E. Kerre. Fuzzy ant based clustering. In *Proceedings of ANTS 2004*, pages 342–349. Springer-Verlag, Berlin, Germany, 2004.
37. A. Vizine, L. de Castro, E. Hruschka, and R. Gudwin. Towards improving clustering ants: An adaptive ant clustering algorithm. *Informatica*, 29(2):143–154, 2005.
38. K.Y. Yeung and W.L. Ruzzo. Details of the Adjusted Rand Index and clustering algorithms. Supplement to the paper "An empirical study on Principal Component Analysis for clustering gene expression data.". *Bioinformatics*, 9(17):763–774, 2001.